REPORT NO: NAWCADPAX--96-196-TR

#103

A SIMPLIFIED FRACTION-FREE INTEGER GAUSS ELIMINATION ALGORITHM

Peter R. Turner, Ph.D.
Mathematics Department
U S Naval Academy
Annapolis, MD 21402

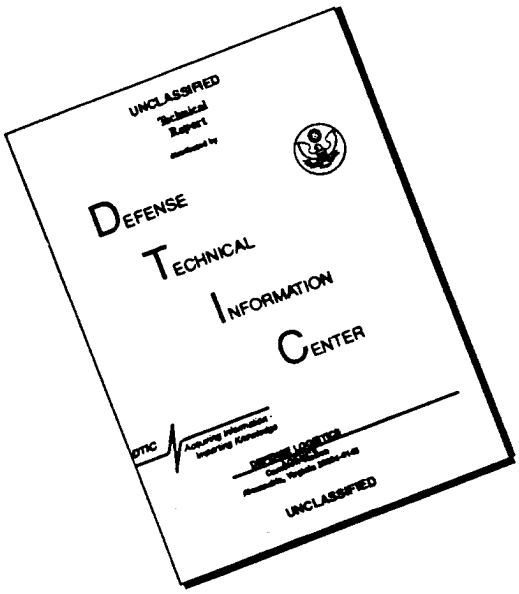
5 AUGUST 1995

FINAL REPORT
Period Covering June 1995 to August 1995

Approved for Public Release; Distribution is Unlimited.

Prepared for
OFFICE OF NAVAL RESEARCH
800 N. Quincy Street
Arlington, VA 22217

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

PRODUCT ENDORSEMENT - The discussion or instructions concerning commercial products herein do not constitute an endorsement by the Government nor do they convey or imply the license or right to use such products.

Reviewed By: _____ **Date:** 7/9/96
Author/COTR

Reviewed By: _____ **Date:** 7/24/96
LEVEL III Manager

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	5 August 1995	June 1995 to August 1995

4. TITLE AND SUBTITLE
A Simplified Fraction-Free Integer Gauss Elimination Algorithm

5. FUNDING NUMBERS

6. AUTHOR(S)
Peter R. Turner, Ph.D.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Air Warfare Center
Aircraft Division Warminster
Code 455100R07
Warminster, PA 18974-0591

8. PERFORMING ORGANIZATION REPORT NUMBER
NAWCADPAX--96-196-TR

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Mathematics Department
U S Naval Academy
Annapolis, MD 21402

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution is Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This paper presents a new version of Gauss elimination for integer arithmetic. This new algorithm allows fraction-free integer computation without requiring any calls to a greatest common divisor routine. It does however keep the growth in the integer dynamic range to a minimum. The algorithm is based on a careful comparison of the divisionless integer GE and the "normal" algorithm using divisions within a floating-point or real arithmetic setting. From this analysis, we identify common factors which are necessarily present throughout the active part of the matrix. These can then be removed by exact integer division. A further consequence of this analysis is that the diagonal entries of the final upper triangular factor are precisely the determinants of the principal minors of the original matrix. In a parallel processing environment, the additional cost of these integer division is minimized since, at each stage, the whole active array is being divided by the same integer.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Fraction-free, Floating-point, Additional cost.	27
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

A Simplified Fraction-Free Integer Gauss Elimination Algorithm

PETER R TURNER
MATHEMATICS DEPARTMENT, U S NAVAL ACADEMY, ANNAPOLIS, MD 21402

ABSTRACT. This paper presents a new version of Gauss elimination for integer arithmetic. This new algorithm allows fraction-free integer computation without requiring any calls to a greatest common divisor routine. It does however keep the growth in the integer dynamic range to a minimum. The algorithm is based on a careful comparison of the divisionless integer GE and the "normal" algorithm using divisions within a floating-point or real arithmetic setting. From this analysis, we identify common factors which are necessarily present throughout the active part of the matrix. These can then be removed by exact integer division. A further consequence of this analysis is that the diagonal entries of the final upper triangular factor are precisely the determinants of the principal minors of the original matrix. In a parallel processing environment, the additional cost of these integer divisions is minimized since, at each stage, the whole active array is being divided by the same integer.

1. INTRODUCTION

In some form Gauss elimination, or *GE* as we shall often abbreviate it here, is probably the most widely used single computational tool in scientific computing for a very wide range of underlying problems. These include solution of partial differential equations, linear programming and least squares approximation in its various guises such as signal processing and linear regression. The basic problems for which it is used are the solution of linear systems of equations (including obtaining the solution space for underdetermined systems), computation of matrix determinants, determination of matrix rank or detection of singularity. Gauss elimination is usually to be found as a major topic in texts on linear algebra (where the emphasis is on its theoretical basis), numerical analysis (with an emphasis on its practical implementation, paying attention to questions of roundoff error and numerical stability), parallel and vector computing (as an important illustration of the potential power of parallel computers). See [1], [3], [5], [10], [11] for examples. However few of these discuss in any detail the interplay between these different perspectives. It is precisely this interplay which leads to the improved integer GE algorithm which is the subject of this paper.

We begin with a brief review of the basic GE algorithm and some variations of this. Section 2 also contains a brief review of the main applications and of the complexity analysis. Section 3 introduces integer GE. The requirements of integer arithmetic raises some different issues. We describe first the division-free form of the algorithm and then the questions raised by this — most notably, the growth of the dynamic range that is needed. Some of these issues were previously discussed for the specific context of residue number systems in [6], [7], [16].

In Section 4, an improved fraction-free integer GE algorithm is developed from a comparison of the matrix entries in the divisionless algorithm and those for the usual GE algorithm. A short discussion of the use of greatest common divisors to reduce the dynamic range growth is also included before the new algorithm is presented. Its use in the various standard applications is also described. Section 5 returns to the question of algorithm complexity and includes some consideration of the demands of (potentially)

long integer wordlength arithmetic. A brief comparison with the use of GE with rational arithmetic is included. The paper concludes with a very short section on extensions of the algorithms presented here to other algebraic settings.

2. THE BASIC ALGORITHMS

2.1. Problem statement and notation. Except where specifically stated we shall consider a square $n \times n$ matrix A although some of the problems and solutions are similarly valid for rectangular matrices. Elements of the matrix A will be denoted by a_{ij}. Its determinant is denoted $\det A$ and its rank by $\mathrm{r}(A)$. The three basic problems we are concerned with here are the solution of systems of equations, computing $\det A$ and determining $\mathrm{r}(A)$. For definitions of any of these, see a standard text such as [1].

In the case of systems of equations, we use the notation

$$A\mathbf{x} = \mathbf{b}$$

and the elements of the unknown and right-hand side vectors are denoted x_i, b_i.

The underlying principle of GE is that multiples of the first equation (or row of the matrix) are subtracted from all subsequent equations (rows) to eliminate the first unknown (element) from each of these. The process is then repeated to eliminate entries below the diagonal of each column in turn. The system of equations is then solved by substitution in the resulting triangular system. Alternatively the determinant of the original matrix is given by the product of the diagonal entries in the resulting matrix. The rank is given by counting the number of nonzero elements on the diagonal of the final array. In the case of solving a system of equations, whatever operations are performed on the matrix must also be performed on the right-hand side.

Of course, these statements are oversimplifications of the true situation but they contain the essence of the approach. In the rest of this section, we present the general version of this simple GE procedure and discuss some of the difficulties that can arise.

2.2. The ijk-form. The "ijk-form" of GE is just the general $n \times n$ version of the algorithm outlined above.

Algorithm 1 Basic GE ijk form

Input $n \times n$ matrix A (and right-hand side $\mathbf{b}$ if solving a system)
Compute
for $i = 1$ to $n-1$
for $j = i+1$ to n
$m := a_{ji}/a_{ii}$
$a_{ji} := 0$
$b_j := b_j - mb_i$ (if solving a system)
for $k = i+1$ to n
$a_{jk} := a_{jk} - ma_{ik}$
Output (modified) matrix A (and $\mathbf{b}$ if solving a system)

Pivoting is the name given to altering the order in which the rows are used in the GE algorithm. The simplest cause for the need for pivoting is the occurrence of a 0 in the appropriate diagonal position so that Algorithm 1 breaks down. In the floating-point environment an almost equally difficult situation is created by a very small diagonal

entry which may be just the result of roundoff error in a quantity which would be 0 if *exact* arithmetic were being used. Such a pivot element can result in large errors in the computed solution. The usual pivoting strategy is *partial pivoting* in which the current i-th column is searched for its element of largest magnitude on or below the diagonal. The row in which this occurs is then interchanged with the i-th row before the elimination continues.

For many purposes, it is desirable to make better use of the work involved in GE. By storing the multipliers used, we obtain the LU factorization of the original matrix A. That is we find a lower triangular factor L and an upper triangular one U such that

$$A = LU$$

or, in the case of pivoting, L, U are factors of a permuted version of the matrix A:

$$PA = LU$$

For this form of the algorithm, the factors L and U are stored in the same locations as the original matrix A. The lower factor L has unit diagonal entries and so these need not be stored explicitly. The principal advantages of the LU factorization lie in the fact that if multiple systems are to be solved with the same coefficient matrix then the factorization can be done just once and each system solved using forward and back substitution loops. This is then *much* cheaper computationally than, for example, inverting the matrix.

For simplicity in the descriptions of the various integer algorithms in the remainder of this paper, we shall restrict our attention to the basic GE algorithm. The inclusion of pivoting is straightforward — though it is *not* obvious what constitutes a good pivoting strategy for integer GE.

There are several variations on the basic ijk-form of GE which have merits for different computing environments. The ijk-form in Algorithm 1 is well-suited to a conventional serial computer architecture with matrices stored by rows. For other storage schemes or processor architectures, alternatives may be preferred. A detailed discussion of these aspects is included in [11]. These variations can take advantage of processors which are designed, for example, for efficient performance of the *saxpy*'s of [5] or of scalar products.

2.3. Applications.

Solving linear systems. The most frequent application of GE and LU factorization is to the solution of a linear system

$$A\mathbf{x} = \mathbf{b} \tag{1}$$

Then Algorithm 1 results in an equivalent upper triangular system

$$U\mathbf{x} = \hat{\mathbf{b}} \tag{2}$$

whose solution is the same as that of (1). This system is then solved using *back substitution.*

Algorithm 2 Back substitution

Input $n \times n$ upper triangular matrix U, n-vector $\mathbf{b}$
Initialize solution $\mathbf{x} := 0$
Compute

```
        for i = n downto 1
            for j = i + 1 to n
                b_i := b_i - u_ij x_j
            x_i := b_i / u_ii
    Output solution x
```

Remark 1. *Note that* $\mathbf{b}$ *has been used (in place of* $\widehat{\mathbf{b}}$*) to denote the right hand side of (2) in this algorithm.*

This version of back substitution has been arranged in such a way that the outer loop results in obtaining one further component of the solution each time. Again there are variations which are better suited to particular architectures.

The computational complexity of floating-point algorithms is often measured by the number of floating-point arithmetic operations that are required. Traditionally, the relative efficiency of algorithms was measured by counting multiplications and divisions and neglecting addition and subtraction operations. For modern processors the time required for floating-point multiplication is not much greater than that for addition and so it is sensible to consider all arithmetic operations. Division still costs substantially more and any elementary function evaluations are typically yet more expensive. The floating-point operation counts for GE are well-known. They can be found in almost any standard text on numerical analysis or computational linear algebra such as [3] or [5].

TABLE 1 Arithmetic operation counts for GE solution of $A\mathbf{x} = \mathbf{b}$.

Operation	Forward elimination	Back substitution	TOTAL
+ or −	$\frac{1}{3}n\left(n^2-1\right)$	$\frac{1}{2}n\left(n-1\right)$	$\frac{1}{6}n\left(n-1\right)\left(2n+5\right)$
×	$\frac{1}{3}n\left(n^2-1\right)$	$\frac{1}{2}n\left(n-1\right)$	$\frac{1}{6}n\left(n-1\right)\left(2n+5\right)$
/	$\frac{1}{2}n\left(n-1\right)$	n	$\frac{1}{2}n\left(n+1\right)$

If multiple systems with the same coefficient matrix are to be solved then the overall operation count for GE is obtained by multiplying these totals by the number of systems. In particular the inversion of the matrix A would therefore result in multiplying all these totals by n so that matrix inversion by GE is an $O(n^4)$ operation. The corresponding table of operation counts for LU factorization makes it easy to see the practical advantage of using the LU factorization whenever multiple systems are to be solved.

Determinant evaluation. Whether we use GE or LU, the evaluation of $\det A$ is extremely simple once the elimination phase is complete. (We are taking no account of any questions of error analysis or stability at this stage.) In the absence of pivoting the determinant is simply the product of the diagonal elements of U.

Rank and singularity detection. Again, neglecting any problems created by roundoff errors or ill-conditioning in the matrix, once GE with pivoting has been completed, singularity is detected simply by seeking a 0 entry in a pivot position. In fact it is sufficient to examine a_{nn} since, if any pivot element is zero, then necessarily $a_{nn} = 0$. Again note we are assuming both pivoting and exact arithmetic. In practice, for floating-point arithmetic at least, this is not sufficient and more care must be taken to test for *near-singularity*. Extending our ideal world analysis, by counting the number of zero pivots we obtain the rank-deficiency of the matrix. Equivalently the number of nonzero pivots yields the rank of A. This is a much more optimistic claim than even the singularity

statement. Within a computer algebra system or with exact arithmetic such statements are valid.

In the domain of real numerical computation, the question of rank determination is more difficult. The effect of roundoff error on GE has been thoroughly analysed. Some of that analysis is summarized in Section 2.5. Packages such as MATLAB[1] (see [9], for example) include rank as a function in their computational library and use a carefully computed tolerance dependent on parameters of the computer system to determine the "true" rank of the original matrix from its Singular Value Decomposition (SVD) [5]. Such aspects are not the focus of this paper.

The solution space for underdetermined systems. In the case of underdetermined systems of equations, whether this is the result of having fewer equations than unknowns or of rank deficiency in the matrix, a set of vectors spanning the solution space is easy to obtain from the LU factorization of the matrix.

The backward substitution must be performed the same number of times as the rank deficiency. Thus if the $n \times n$ matrix has rank r, we solve the smaller system $n - r$ times each time with r of the unknowns specified. To guarantee linearly independent solutions, it suffices to use the standard basis vectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_{n-r}$ in turn to specify the values of $x_{r+1}, x_{r+2}, \ldots, x_n$. That is, we first set $(x_{r+1}, x_{r+2}, \ldots, x_n) = (1, 0, \ldots, 0)$ and solve for the remaining unknowns. The process is repeated for $(x_{r+1}, x_{r+2}, \ldots, x_n) = (0, 1, 0, \ldots, 0)$ and so on until we have used $(x_{r+1}, x_{r+2}, \ldots, x_n) = (0, \ldots, 0, 1)$.

The algorithms are simple extensions of those outlined earlier. Similar basis vectors could be used in more abstract settings with symbolic computer algebra systems.

3. GAUSS ELIMINATION OVER THE INTEGERS

For computation over the integers, it is necessary to make changes to the basic GE algorithms described in Section 2. The most obvious cause is the fact that the integers are not closed under division. This has the side effect that the magnitudes of integers generated during the computation can grow rapidly. The range of integer values required or available is known as the *dynamic range*. Overflowing the integer range in binary integer arithmetic often results in the phenomenon known as "integer wraparound" (see [3], Chapter 1, for example) which is the effect of the "clock" arithmetic modulo 2^N where N is the integer wordlength in bits. Further complications that arise out of integer arithmetic (however it is performed) include choosing a good pivoting strategy.

In a Computer Algebra System (CAS) such as Maple[2] [2] or Mathematica[3] [20], some of these problems are avoided at the expense of computational speed since exact arithmetic with very long integers can be performed in software. However such computation becomes very slow when the arithmetic wordlength gets long — and the rate of growth of the dynamic range can be very rapid.

One way of overcoming some of the difficulties is the use of alternative representation and arithmetic formats for the integers. The residue number systems RNS are well-suited to some of these tasks. In such a system, an integer is represented by its residues modulo a number of different prime numbers. The advantage here is that the growth of the dynamic range is achieved by extending the set of basis primes being used. RNS arithmetic has a natural short wordlength parallelism which avoids the slowdown caused by very long

[1] MATLAB is a registered trademark of The MathWorks, Inc

[2] Maple is a registered trademark of Maple Waterloo Software, Inc.

[3] Mathematica is a registered trademark of Wolfram Research, Inc.

wordlength arithmetic. However it brings with it other difficulties which are less easily resolved. Even if one integer divides another and both are within the dynamic range of the system, there is no simple division algorithm which returns this result; range checking is (at best) very difficult; comparison is not an RNS operation. RNS arithmetic systems and processors have been extensively studied; see [13] or [15] for example.

In this section, we discuss the implementation and use of GE using integer arithmetic. Our primary focus will be on binary integer arithmetic. The use of RNS arithmetic within the specific context of adaptive beamforming was discussed in [6], [16]. The latter reference addresses the issues of dynamic range and complexity in this setting.

3.1. Division-free Gauss elimination. The simplest way of modifying GE to integer arithmetic is to eliminate the divisions by performing "cross-multiplications" between the rows of the matrix. This is the form which generates the greatest rate of growth in the dynamic range. This corresponds to the transformation of 2×2 matrices $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ to $\begin{bmatrix} a & b \\ 0 & ad-bc \end{bmatrix}$ instead of $\begin{bmatrix} a & b \\ 0 & d-b(c/a) \end{bmatrix}$. This is achieved by multiplying the second row by a, and subtracting from it c times the first one. The overall divisionless GE algorithm consists of repeating this for all the appropriate submatrices as in the basic form of the algorithm in Section 2.

Algorithm 3 Division-free GE ijk form

Input $n \times n$ integer matrix A (and right-hand side b if solving a system)
Compute
for $i = 1$ to $n-1$
for $j = i+1$ to n
$b_j := a_{ii}b_j - a_{ji}b_i$ (if solving a system)
for $k = i+1$ to n
$a_{jk} := a_{ii}a_{jk} - a_{ji}a_{ik}$
$a_{ji} := 0$
Output (modified) matrix A (and b if solving a system)

Depending on the individual task being performed, this elimination algorithm would then be followed with the appropriate final stages — a modified back substitution for solving a system or other modifications of the algorithms of Section 2 for rank determination or determinant calculation. These are discussed in Section 4 in the context of the new version of the algorithm.

It is easy to see from the 2×2 situation that the matrix elements are likely to grow rapidly during this process. By way of contrast Wilkinson [19] establishes that there is no growth in the dynamic range if partial pivoting is used with Algorithm 1.

3.2. Growth in dynamic range. The question of the range growth in divisionless GE was addressed in some detail in [6], [7], [16] in order to analyse the possibilities for RNS arithmetic within the context of adaptive beamforming. In this section we begin by summarizing these results for general integer arithmetic. In order to develop satisfactory algorithms for the various underlying problems, it is also useful to examine the relation between the matrix entries arising from the divisionless algorithm and the corresponding algorithm using division. This question is addressed later in this subsection.

To get a feel for the potential range growth in the divisionless GE algorithm, consider first just a 2×2 matrix with integer entries in the range $[-M, M]$. Usual binary integer representations have a range of the form $[-M, M-1]$; but for the present purpose, we remark that the initial range does not necessarily match the available dynamic range. The symmetric range simplifies the analysis of the growth.

The transformation of the 2×2 matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ to $\begin{bmatrix} a & b \\ 0 & ad-bc \end{bmatrix}$ results in an element $ad - bc \in [-2M^2, M^2]$. If $M = 2^K - 1$, this implies that the dynamic range required has increased from a required minimum wordlength of $K+1$ bits to $2K+2$ bits. This same exponential rate of growth is possible at every stage of the outer loop of Algorithm 3.

Thus the initial required wordlength is (approximately) doubled $N-1$ times during the divisionless GE elimination for an $N \times N$ matrix. The final wordlength needed is therefore around $2^{N-1}(K+1)$. It is easy to see that this would very quickly exhaust any normally available dynamic range.

For example, if K were just 3 so that the initial integer range is restricted to just $[-7, 7]$ with $N = 6$, the final dynamic range would need a wordlength of at least $2^5 \times 4 = 128$ bits which is already double the length of any commonly found built-in integer format. Clearly for a more realistic range for the initial data and larger matrix dimensions this growth would very quickly exceed all plausible ranges even for software implementation.

In the case of complex integer arithmetic as in [7] the growth is even slightly more rapid than the above analysis suggests and, perhaps more importantly, the worst case growth is achieved in realistic examples. Clearly it becomes necessary to find ways of restricting this growth. The standard approach is to search for greatest common divisors and to factor them out of subsequent computation.

3.3. Reducing the range growth. In the next section, we consider how to take full advantage of common factors which result from the divisionless algorithm. In this section we look first at the simplest range reduction technique resulting from the removal of unnecessary factors.

Greatest common divisors. The simplest approach is just to divide out any common factors in the "cross-multiplication" operations which are at the heart of the divisionless GE algorithm. Again, this idea is easily understood by considering the 2×2 elimination beginning with $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. Suppose that a, c have a common factor p so that there exist integers α, γ such that $a = p\alpha$, $c = p\gamma$. Then the usual divisionless transformation to $\begin{bmatrix} a & b \\ 0 & ad-bc \end{bmatrix}$ can be replaced by subtracting γ times the first row from α times the second to yield $\begin{bmatrix} a & b \\ 0 & \alpha d - b\gamma \end{bmatrix}$ since $\alpha c - a\gamma = \alpha p\gamma - p\alpha\gamma = 0$.

This suggests that it is desirable to find the *greatest common divisor*, $\gcd(a, c)$, of a, c before proceeding with the elimination. This same principle can be applied at each step of the process and it can be applied in more general settings than just integer arithmetic. For example, this idea is incorporated into the polynomial fraction-free Gauss elimination functions for Maple.

Further simplification may be achievable using the gcd in the (perhaps unlikely) event that a complete row of the matrix at some stage has a nontrivial common factor. This too can be divided out, reducing the range growth in subsequent stages of the elimination.

Of course, if the goal is to obtain $\det A$ then a record of (the product of) these factors must be kept to multiply the final result. Finding the gcd of a complete row of a matrix of anything more than very small dimension may be at a price which does not justify the savings. The simplest technique for finding $\gcd(a, c)$ is the Euclidean algorithm which is easily implemented as follows:

Algorithm 4 Euclidean algorithm for integer gcd

Input Positive integers $c < a$
Initialize $q := c$, $tmpc := a$
Repeat
$tmpa := tmpc$; $tmpc := q$
$q := tmpa \bmod tmpc$
until $q := 0$
Output $\gcd(a, c)$ is $tmpc$

This can be used within the following modified divisionless GE algorithm.

Fraction-free algorithm. A fraction-free version of GE can be defined using the gcd algorithm above to reduce the range growth effect. The resulting algorithm needs no divisions beyond the removal of known factors.

Algorithm 5 Fraction-free GE using greatest common divisors ijk form

Input $n \times n$ integer matrix A (and right-hand side $\mathbf{b}$ if solving a system)
Compute
for $i = 1$ to $n - 1$
for $j = i + 1$ to n
$p := \gcd(a_{ii}, a_{ji})$
$\alpha := a_{ii}/p$; $\gamma := a_{ji}/p$
$b_j := \alpha b_j - \gamma b_i$ (if solving a system)
for $k = i + 1$ to n
$a_{jk} := \alpha a_{jk} - \gamma a_{ik}$
$a_{ji} := 0$
Output (modified) matrix A (and $\mathbf{b}$ if solving a system)

Since we have no advance knowledge of the existence of nontrivial factors (that is $p > 1$) this algorithm does not obviously moderate the worst case range growth of the above analysis. In fact the comparative analysis of the next section suggests that it would have some beneficial effect — but at considerably greater cost than is necessary.

Using the gcd of complete rows can be incorporated into the algorithm at any stage. Potentially such factors could exist in any row of the active matrix and such factors could be sought in every row at every stage of the elimination. The algorithmic changes are straightforward but the cost is likely to be too high and so we do not elucidate further here — except for one very important special situation which does arise as a result of the elimination process itself.

4. THE NEW ALGORITHM

The development of the new fraction-free integer GE algorithm is based on a comparison between the matrix entries which result with or without divisions. The most important result of this analysis is that certain common factors are found to be generated by the

algorithm in a completely predictable way. These can be removed using exact integer division. The range growth is thus restricted to a level which may be considered inherent in the problem.

4.1. Comparison of real and integer Gauss elimination. In this subsection, we look in some detail at the relations between the matrix elements generated by the divisionless form Algorithm 3 of GE and those that would be obtained with divisions as in Algorithm 1. This is important for the completion of the various applications especially computing the determinant which was especially straightforward for Algorithm 1. In order to make this comparison, we assume that the divisions can be performed exactly in some rational arithmetic system. Since these divisions only occur within an entirely theoretical version of the algorithm, the comparisons remain valid and useful.

In [7] it is stated that the growth of the matrix elements is such that the final value of a_{NN} at the conclusion of the elimination phase is $\det A$. This turns out to be overoptimistic in general. The potential growth is much greater than this although it becomes apparent from the analysis that the algorithm can be modified to have this result. First we need some notation to distinguish between elements resulting from the different forms of GE under consideration. For simplicity here we shall completely ignore any pivoting and shall assume that the algorithm does not break down due to a zero pivot.

Following convention, we denote by a_{ij} the elements of the original matrix. All reference will be to the simplest ijk-form of GE described by Algorithm 1. By the i-th *stage* of the algorithm we mean the outermost loop for the value i which performs elimination below the diagonal in the i-th column. The entries resulting from the i-th GE with divisions are denoted by $a_{jk}^{(i)}$. The corresponding entries for the divisionless form Algorithm 3 will be denoted by $b_{jk}^{(i)}$ $(j,k>i)$. The "final" values in the two algorithms are therefore given by $a_{jk}^{(j-1)}$, $b_{jk}^{(j-1)}$ $(j=0,1,\ldots,N-1;\ k\geq j)$ where $a_{1k}^{(0)}=b_{1k}^{(0)}=a_{1k}$.

To see the relations between the entries $b_{jk}^{(i)}$ and $a_{jk}^{(i)}$, we look first at the transformation of $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ to $\begin{bmatrix} a & b \\ 0 & ad-bc \end{bmatrix}$ as opposed to $\begin{bmatrix} a & b \\ 0 & d-b(c/a) \end{bmatrix}$. It is easy to see that, in the current notation,

$$b_{22}^{(1)} = ad - bc = a\,[d - b(c/a)] = a_{11}a_{22}^{(1)}$$

and that the corresponding relation would hold for all other matrix entries resulting from the first stage. That is

$$b_{jk}^{(1)} = a_{11}a_{jk}^{(1)} \qquad (2\leq j,k\leq N) \tag{3}$$

In particular, we see that $b_{22}^{(1)}$ is the determinant of the top-left 2×2 submatrix. It will be convenient to denote this by d_2. In general, we shall denote by d_i the determinant of the top-left $i\times i$ submatrix of A.

The next stage of the elimination, is essentially the same as this first stage operating on the bottom-right $(N-1)\times(N-1)$ square submatrix — the "active matrix". It follows that there is a further scaling of all affected entries by the pivot element $b_{22}^{(1)}$. Thus, we deduce that

$$b_{jk}^{(2)} = a_{11}b_{22}^{(1)}a_{jk}^{(2)} \qquad (3\leq j,k\leq N) \tag{4}$$

and continuing in this way we obtain the general relation:

$$b_{jk}^{(i)} = a_{11}b_{22}^{(1)}\cdots b_{ii}^{(i-1)}a_{jk}^{(i)} \qquad (i<j,k\leq N) \tag{5}$$

Each "final" divisionless entry is its corresponding value from Algorithm 1 scaled by the product of all the final diagonal entries *of the divisionless algorithm* above it.

We note immediately that this analysis suggests that entries in the active matrix have common factors which have been included by the algorithm itself. These represent one obvious way of reducing the range growth and altering the algorithm. This modification, which is presented below, will result in each diagonal element being the determinant d_i of the principal minor of the appropriate dimension.

4.2. A new fraction-free GE algorithm. From (4) it follows, in particular, that

$$b_{33}^{(2)} = a_{11} b_{22}^{(1)} a_{33}^{(2)} = a_{11} \left(a_{11} a_{22}^{(1)} \right) a_{33}^{(2)} = a_{11} d_3$$

where, we recall, $d_3 = a_{11} a_{22}^{(1)} a_{33}^{(2)}$ is the determinant of the principal 3×3 minor. We remark that d_3 is an integer. It follows that $b_{33}^{(2)}$ has the factor a_{11}. Using this same reasoning, we see that $b_{jk}^{(2)}$ has a factor a_{11} for every $j, k \geq 3$ since each such element is a_{11} times the (integer) determinant of a 3×3 minor. This known factor can easily be removed prior to the next stage of the elimination.

Similar factors are introduced into the active matrix at each subsequent stage. These too can be divided out. The factors introduced at the subsequent stages of the elimination are the (modified) diagonal entries. Since these are known factors, there is no need for any calls to a gcd function.

The removal of these factors has an obvious effect on the range growth in subsequent stages of the elimination even though its effect on the worst case analysis is unclear. The growth in the required wordlength could be computed "on-the-fly" in order to take full advantage of this saving.

The division by these factors is incorporated into Algorithm 6 below. We observe that this algorithm, like Algorithm 5, is fraction-free but not division-free. All divisions that are performed are *integer operations with exact integer results.*

Algorithm 6 Fraction-free GE ijk form

Input $n \times n$ integer matrix A (and right-hand side **b** if solving a system)
Compute

```
for i = 1 to n − 1
    for j = i + 1 to n
        b_j := a_ii b_j − a_ji b_i        (if solving a system)
        for k = i + 1 to n
            a_jk := a_ii a_jk − a_ji a_ik
        a_ji := 0
    if i ≥ 2 then                (removal of common factors)
        for j = i + 1 to n
        b_j := b_j / a_{i−1,i−1}          (if solving a system)
            for k = i + 1 to n
                a_jk := a_jk / a_{i−1,i−1}
```

Output (modified) matrix A (and **b** if solving a system)

To gain some insight into the saving that results from this algorithm, consider just the final value of a_{NN}. In the division-free Algorithm 3, using (5) for $i = N - 1$, we have:

$$b_{NN}^{(N-1)} = a_{11} b_{22}^{(1)} \cdots b_{N-1,N-1}^{(N-2)} a_{NN}^{(N-1)}$$

which in turn yields

$$\begin{aligned} b_{NN}^{(N-1)} &= a_{11}a_{22}^{(1)}\cdots a_{N-1,N-1}^{(N-2)}a_{NN}^{(N-1)}\left\{a_{11}^{N-2}\left[b_{22}^{(1)}\right]^{N-3}\cdots b_{N-2,N-2}^{(N-3)}\right\} \\ &= d_N\left\{a_{11}^{N-2}\left[b_{22}^{(1)}\right]^{N-3}\cdots b_{N-2,N-2}^{(N-3)}\right\} \end{aligned} \tag{6}$$

The corresponding final value for Algorithm 6 is just

$$a_{NN} = \det A = d_N \tag{7}$$

which is typically very much smaller since these additional factors have been removed during the modified elimination. Indeed it turns out for this Algorithm 6 that at the conclusion of the elimination we have

$$a_{ii} = d_i. \tag{8}$$

Example 1. *Comparison of Algorithms 3 and 6 for a 4 × 4 matrix. The results of exact arithmetic in Algorithm 1 are also included for comparison and illustration.*

Let

$$A = \begin{bmatrix} 8 & 7 & 4 & 1 \\ 4 & 6 & 7 & 3 \\ 6 & 3 & 4 & 6 \\ 4 & 5 & 8 & 2 \end{bmatrix}$$

Algorithm 1 yields the following sequence of modified matrices:

$$\begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 5/2 & 5 & 5/2 \\ 0 & -9/4 & 1 & 21/4 \\ 0 & 3/2 & 6 & 3/2 \end{bmatrix} \quad \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 5/2 & 5 & 5/2 \\ 0 & 0 & 11/2 & 15/2 \\ 0 & 0 & 3 & 0 \end{bmatrix} \quad \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 5/2 & 5 & 5/2 \\ 0 & 0 & 11/2 & 15/2 \\ 0 & 0 & 0 & -45/11 \end{bmatrix}$$

from which we may deduce that $\det A = (8)(5/2)(11/2)(-45/11) = -450$.

The division-free Algorithm 3 gives:

$$\begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & -18 & 8 & 42 \\ 0 & 12 & 48 & 12 \end{bmatrix} \quad \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & 0 & 880 & 1200 \\ 0 & 0 & 480 & 0 \end{bmatrix} \quad \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & 0 & 880 & 1200 \\ 0 & 0 & 0 & -576000 \end{bmatrix}$$

which shows the very rapid growth which is possible with this algorithm.

The fraction-free Algorithm 6 also gives

$$\begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & -18 & 8 & 42 \\ 0 & 12 & 48 & 12 \end{bmatrix} \quad \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & 0 & 880 & 1200 \\ 0 & 0 & 480 & 0 \end{bmatrix}$$

but the active part of this matrix is then divided by the common factor 8 to yield

$$\begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & 0 & 110 & 150 \\ 0 & 0 & 60 & 0 \end{bmatrix}$$

which in turn gives

$$\begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & 0 & 110 & 150 \\ 0 & 0 & 0 & -9000 \end{bmatrix}$$

The active matrix is now just the bottom-right element which needs to be divided by the "common factor" 20 to yield:

$$\begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 20 & 40 & 20 \\ 0 & 0 & 110 & 150 \\ 0 & 0 & 0 & -450 \end{bmatrix}$$

The final values of the diagonal entries can easily be seen (by comparison with the partial products of the diagonal of the final upper triangle generated by Algorithm 1) to be the determinants of the appropriate principal minors, as predicted by (7) and (8).

Although there has been some growth in the magnitudes of the matrix elements it has been kept in much better check. This growth is no greater than that which must be accommodated if the determinant of the original matrix is to be computed. The whole of this computation could have been achieved using standard 16-bit integer arithmetic - even including the temporary values. The division-free algorithm in this case requires at least 20 bits even though the original matrix has integer elements bounded by 8.

The benefits in terms of range growth derived from using Algorithm 6 in the above example would be essentially matched by using Algorithm 5 in this case but the latter algorithm would also require 6 gcd operations. For Algorithm 6, the factors to be removed are known and automatically stored as part of the matrix. The sequence of matrices generated by Algorithm 5 for this example is

$$\begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 5 & 10 & 5 \\ 0 & -9 & 4 & 21 \\ 0 & 3 & 12 & 3 \end{bmatrix} \quad \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 5 & 10 & 5 \\ 0 & 0 & 110 & 150 \\ 0 & 0 & 30 & 0 \end{bmatrix} \quad \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 5 & 10 & 5 \\ 0 & 0 & 110 & 150 \\ 0 & 0 & 0 & -450 \end{bmatrix}$$

We should observe however that it is *not* generally the case that Algorithm 5 yields $a_{NN} = d_N$ nor that it restricts the growth in the dynamic range as effectively as this. Note that the original matrix has a common factor of 2 throughout its first column.

Removal of common factors from rows of the active matrix for the above example would yield

$$\begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 1 & 2 & 1 \\ 0 & -9 & 4 & 21 \\ 0 & 1 & 4 & 1 \end{bmatrix} \quad \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 11 & 15 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 8 & 7 & 4 & 1 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 11 & 15 \\ 0 & 0 & 0 & -15 \end{bmatrix}$$

which clearly has a greatly beneficial effect on the dynamic range — but at a considerable cost. There are 22 gcd operations required in order to achieve this saving. If the $\det A$ is required it is not a simple matter to recover it from the resulting upper triangular factor even with a knowledge of the factors which have been removed. If a linear system were to be solved, the number of gcd operations increases yet further with no guarantee of retaining the savings seen here.

It is informative to consider a second example which is just a column permutation of this first one.

Example 2. *We repeat much of Example 1 for the matrix*

$$A = \begin{bmatrix} 7 & 4 & 1 & 8 \\ 6 & 7 & 3 & 4 \\ 3 & 4 & 6 & 6 \\ 5 & 8 & 2 & 4 \end{bmatrix}$$

which is permuted so that there is no convenient common factor in the first column.

The division-free Algorithm 3 gives:

$$\begin{bmatrix} 7 & 4 & 1 & 8 \\ 0 & 25 & 15 & -20 \\ 0 & 16 & 39 & 18 \\ 0 & 36 & 9 & -12 \end{bmatrix} \quad \begin{bmatrix} 7 & 4 & 1 & 8 \\ 0 & 25 & 15 & -20 \\ 0 & 0 & 735 & 770 \\ 0 & 0 & -315 & 420 \end{bmatrix} \quad \begin{bmatrix} 7 & 4 & 1 & 8 \\ 0 & 25 & 15 & -20 \\ 0 & 0 & 735 & 770 \\ 0 & 0 & 0 & 551250 \end{bmatrix}$$

which again shows the expected very rapid growth.

The corresponding sequence of matrices generated by the fraction-free Algorithm 6 is

$$\begin{bmatrix} 7 & 4 & 1 & 8 \\ 0 & 25 & 15 & -20 \\ 0 & 16 & 39 & 18 \\ 0 & 36 & 9 & -12 \end{bmatrix} \quad \begin{bmatrix} 7 & 4 & 1 & 8 \\ 0 & 25 & 15 & -20 \\ 0 & 0 & 105 & 110 \\ 0 & 0 & -45 & 60 \end{bmatrix} \quad \begin{bmatrix} 7 & 4 & 1 & 8 \\ 0 & 25 & 15 & -20 \\ 0 & 0 & 105 & 110 \\ 0 & 0 & 0 & 450 \end{bmatrix}$$

in which we see not only that the final value of a_{44} is the determinant of the original matrix but that this matrix is positive definite since all its principal minors have positive determinants.

In this case using Algorithm 5 so that common factors are removed form the multipliers, we get

$$\begin{bmatrix} 7 & 4 & 1 & 8 \\ 0 & 25 & 15 & -20 \\ 0 & 16 & 39 & 18 \\ 0 & 36 & 9 & -12 \end{bmatrix} \quad \begin{bmatrix} 7 & 4 & 1 & 8 \\ 0 & 25 & 15 & -20 \\ 0 & 0 & 735 & 770 \\ 0 & 0 & -315 & 420 \end{bmatrix} \quad \begin{bmatrix} 7 & 4 & 1 & 8 \\ 0 & 25 & 15 & -20 \\ 0 & 0 & 735 & 770 \\ 0 & 0 & 0 & 15750 \end{bmatrix}$$

by "removal" of the common factor 35 from 735 and -315 in the multipliers used for the last step. Clearly this time the range reduction achieved by Algorithm 5 is much inferior to that of the new algorithm.

Removal of common factors from rows of the active matrix at every stage yields

$$\begin{bmatrix} 7 & 4 & 1 & 8 \\ 0 & 5 & 3 & -4 \\ 0 & 16 & 39 & 18 \\ 0 & 12 & 3 & -4 \end{bmatrix} \quad \begin{bmatrix} 7 & 4 & 1 & 8 \\ 0 & 5 & 3 & -4 \\ 0 & 0 & 21 & 22 \\ 0 & 0 & -3 & 4 \end{bmatrix} \quad \begin{bmatrix} 7 & 4 & 1 & 8 \\ 0 & 5 & 3 & -4 \\ 0 & 0 & 21 & 22 \\ 0 & 0 & 0 & 150 \end{bmatrix}$$

The beneficial effect on the dynamic range is again apparent but the disadvantages noted after Example 1 remain.

4.3. Applications. In this section, we describe the effect of the modified fraction-free GE Algorithm 6 on the solution of the various underlying problems. From the observations in the last section, it is plain that the evaluation of $\det A$ is particularly simple, provided only that no zero pivots arise during the computation.

Linear systems. For the solution of a (nonsingular) linear system using Algorithm 6, there is of course no guarantee that the solution vector has only integer components. If the system is known (because of the context, perhaps) to have an integer solution then this can be computed by applying the conventional back substitution (Algorithm 7 or Algorithm 8, for example) in which case the divisions will again have integer results. Otherwise the solution can be expressed as fractions using rational arithmetic with the obvious modifications of Algorithm 2.

Example 3. *Consider the coefficient matrix of Example 1 with the original right-hand side vector* $[45, 30, 40, 30]^{\mathrm{T}}$.

Algorithm 6 would generate the augmented matrix

$$\left[\begin{array}{cccc|c} 8 & 7 & 4 & 1 & 45 \\ 0 & 20 & 40 & 20 & 60 \\ 0 & 0 & 110 & 150 & 260 \\ 0 & 0 & 0 & -450 & -450 \end{array}\right]$$

from which we obtain the integer solutions as $x_4 = (-450)/(-450) = 1$, so that $x_3 = [260 - 150(1)]/110 = 1$. Then $x_2 = [60 - 40(1) - 20(1)]/20 = 0$ and $x_1 = (45 - 4 - 1)/8 = 5$.

If we did not know in advance that the original system has an integer solution vector, rational arithmetic would be used to generate the equivalent results.

Determinant. We have already seen that Algorithm 6 delivers $\det A$ automatically as the final value of a_{NN}. Furthermore, since the final diagonal consists of the determinants of the principal minors of increasing dimension, positive or negative (semi-) definiteness can also be detected simply and automatically.

Rank determination. In the absence of a zero pivot, again there is nothing further to be done. In the event of such a zero, then interchanging the pivot row with any lower row with a nonzero entry in the pivot column will allow the fraction-free algorithm to proceed. Simply counting the number of nonzero entries on the diagonal gives the rank, as before. This simple interchange is not necessarily the optimal pivoting strategy for integer computation.

In the event of a rank-deficient matrix, obtaining the solution space for an underdetermined system can be accomplished by modifying the back substitution algorithm in just the same way as for real-number computation.

Pivoting. Clearly in the event of a zero pivot some pivoting is necessary in any good implementation of GE. The question is what is the right strategy for integer computing? Choosing the largest element in the floating-point algorithm has the virtue of keeping all multipliers small and therefore restricting the growth of the matrix elements. However that restricted growth is only realized because of the divisions.

At least intuitively, choosing the largest element in the pivot column is not necessarily good for integer computation. Indeed a large prime pivot is probably near-worst since that appears to almost guarantee rapid growth in the dynamic range. In (4) and then (5)

we see that each $b_{jk}^{(2)}$ has the factor a_{11} and each further stage has this factor and then has it repeated in the factors $b_{ii}^{(i-1)}$. The earlier a particular pivot is used the greater the impact it has on subsequent range growth. This is made plain in equation (6).

In the fraction-free version Algorithm 6, however, the range growth is essentially independent of the order of the use of the rows. In the case of a full rank matrix A, the final entry $a_{NN} = d_N$ as in (7) is often the largest of the principal minor determinants and this value is invariant (up to sign changes) under row interchanges. It follows that the simplest pivoting strategy is probably the best for this algorithm. That is, at stage i, we should use the first row for which the pivot column entry is nonzero: choose $pivot = \min\{p \geq i : a_{pi} \neq 0\}$.

It is conceivable that a different ordering may represent some minor improvement on this. For example, looking for pivots which are either powers of the binary (or other computational) base may make subsequent divisions particularly simple. However, searching for the appropriate pivot element in this regard would (almost surely) be more wasteful than beneficial.

5. COMPLEXITY OF THE FRACTION-FREE ALGORITHM

In this section, we begin by obtaining the basic integer operation counts for the fraction-free GE Algorithm 6. There are two essential differences between these counts and those for the floating-point algorithms: the fundamental elimination loops contain no divisions but have twice as many multiplications, and there is the additional complexity of the removal of the common factors after the first two stages so that divisions occur at a slightly later stage.

The additional multiplications in the elimination are easily counted. The common factor removal entails only divisions. The total number of these can be assessed from the loop control limits. For each $i \geq 2$, there is one division in the j-loop itself (assuming we are solving a system) which runs from $j = i+1$ to n. Also in this loop is the k-loop which has the same limits and contains another division. The total division count is therefore

$$\sum_{i=2}^{n-1}(n-i)(n-i+1) = \sum_{l=1}^{n-2} l(l+1) = \frac{1}{3}n(n-1)(n-2)$$

The total operation counts are summarized in Table 4.

TABLE 4 Integer arithmetic operation counts for the fraction-free GE Algorithm 6 for solution of an $n \times n$ linear system $A\mathbf{x} = \mathbf{b}$.

Operation	Matrix Factorization	Right-hand side	TOTAL
+ or −	$\frac{1}{3}n\left(n^2-1\right)$	$\frac{1}{2}n\left(n-1\right)$	$\frac{1}{6}n\left(n-1\right)\left(2n+5\right)$
×	$\frac{2}{3}n\left(n^2-1\right)$	$n\left(n-1\right)$	$\frac{1}{3}n\left(n-1\right)\left(2n+5\right)$
/	$\frac{1}{6}(n-2)\left(n-1\right)\left(2n-3\right)$	$\frac{1}{2}(n-1)(n-2)$	$\frac{1}{3}n\left(n-1\right)\left(n-2\right)$

Note that this operation count does *not* include the back substitution. What we see most importantly is that the number of multiplications has doubled *and*, even worse, the number of divisions has increased by a complete order from $O(n^2)$ to $O(n^3)$.

Unfortunately this is not the end of the story. The complexity of this algorithm is further increased by virtue of the fact that these integer divisions are typically more difficult than their floating-point counterparts — especially with the range growth which we have already seen can be substantial. This is likely to necessitate the use of *long* integer arithmetic for which special algorithms must be used.

5.1. Long integer arithmetic. Long integer arithmetic can be simulated using multiple words of some basic wordlength.

For example, if the underlying integer wordlength is 8 bits (or 1 byte) then such an integer can be regarded as a radix $2^8 = 256$ digit. Conventionally the range of values of the base 256 digits would be $-128, -127, \ldots, 127$ so that signed integers can be represented. For simplicity in the current description, we restrict our attention to nonnegative integers with a digit range of $0, 1, \ldots, 255$. Very large integers could then be stored using a vector of such integers using conventional place value.

In general, suppose the base wordlength is L bits so that the effective radix is $R = 2^L$. The vector $(d_0, d_1, \ldots, d_{K-1})$ would then be used to represent the integer $N \in [0, R^K - 1]$ given by

$$N = d_{K-1}R^{K-1} + d_{K-2}R^{K-2} + \cdots + d_1 R + d_0 \tag{9}$$

where each digit satisfies $0 \leq d_i < R$. For efficient arithmetic using such a representation we require an integer accumulator with $2L$ bits. Among other consequences of this are that addition can be computed in "digit-parallel" with subsequent attention to any carries which may be propagated. A large radix carry lookahead or conditional sum adder could be constructed to improve the efficiency of addition. These addition algorithms are simple generalizations of the usual binary addition algorithms which are described, for example, in [12].

Multiplication of long integers can be achieved with reasonable efficiency using the convolution form of the product working with the component words of the large integers. Again for simplicity, we shall only consider multiplication of positive integers. The multiplication of two integers in the form (9) will result in an integer whose representation requires at most $2K$ L-bit words.

Suppose that we require the product of the long integers

$$m = \sum_{i=0}^{K-1} \alpha_i R^i, \qquad n = \sum_{i=0}^{K-1} \beta_i R^i$$

This product is given by

$$m * n = \left(\sum_{i=0}^{K-1} \alpha_i R^i\right)\left(\sum_{j=0}^{K-1} \beta_j R^j\right) = \sum_{k=0}^{2K-2}\left(\sum_{i=0}^{K-1} \alpha_i \beta_{k-i}\right) R^i \tag{10}$$

where we use the convention $\beta_j = 0$ if $j < 0$. This is essentially a convolution product of the coefficient vectors.

Now each coefficient product $\alpha_i \beta_{k-i}$ can be viewed as a base-R^2 digit which we can write in the form $\alpha_i \beta_{k-i} = \gamma_{k,i} R + \delta_{k,i}$ where each $\gamma_{k,i}$ and $\delta_{k,i}$ is a base-R digit. Thus $\gamma_{k,i}$ is the most significant and $\delta_{k,i}$ the least significant R-digit of $\alpha_i \beta_{k-i}$. The product (10) can therefore be written as

$$m * n = \sum_{k=0}^{2K-2}\left(\sum_{i=0}^{K-1} \gamma_{k,i} R + \delta_{k,i}\right) R^i = \sum_{k=0}^{2K-1}\left(\sum_{i=0}^{K-1} \gamma_{k-1,i} + \delta_{k,i}\right) R^i \tag{11}$$

where $\gamma_{k,i} = \delta_{k,i} = 0$ for $i > k$.

Remark 2. *Carries beyond the R^{2K-1} position are not possible since $m*n \leq (R^K - 1)^2 < R^{2K}$.*

Remark 3. *The results of the inner sums in (11) will usually create further carries which must be accounted for. However, we note that for almost any minimal degree of parallelism in the processor, each of these inner sums can be performed simultaneously since they consist of at most $2K$ terms each less than R and we would expect $2K \ll R$. Therefore the sum will be less than R^2 so that it will be representable in a double length accumulator.*

For the 8-bit basic wordlength the restriction is only that our integers do not exceed $(2^8)^{256} = 2^{2048} > 10^{616}$ which is well beyond any typical integer computing range for linear algebra problems.

The length of the inner sums in (11) will also restrict the size of any carry and therefore may be useful in bounding the range of the propagation of such carries. This could be used to improve the efficiency of such a convolution product. We do not consider such details further in this report.

What effect does such a long multiplication have on the arithmetic complexity of an integer algorithm? The product formula (10) entails K^2 basic multiplications. These components must then be broken into their two component digits and then approximately K^2 additions together with the carries these generate. In Table 4, this means that each of the (approximately) $\frac{2}{3}n^3$ multiplications entails something like $2K^2$ regular integer arithmetic operations. For even quite moderate range growth with $K = 4$ this has the effect of increasing this part of the complexity by a factor of 32.

We conclude this section with a brief comment on the impact of parallelism on performing Gauss elimination using integer computation and on the long integer arithmetic required for the growth in the dynamic range. An array processor could be used to accelerate this algorithm greatly. Firstly, all the coefficient products in the inner sum in (10) can be computed simultaneously. The realignment of the upper and lower halves of these products needed for the inner summation in (11) is then a simple shift of data to a neighboring processor and these sums could then also be performed simultaneously. The final carries would be the only part that requires serial processing.

Division of long integers can also be achieved by generalizing some of the standard algorithms which are used in binary integer hardware such as the SRT division algorithm which is based on the idea of nonrestoring division. This algorithm is well-suited to high-radix division and so can be modified to the long integer framework. The SRT algorithm relies on repeated addition and subtraction using signed digits. Since we have only dealt with arithmetic of nonnegative long integers here, we do not discuss the detailed implementation further. For details of the basic algorithm and its implementation at least for radix 4, see [12].

5.2. Computing with the rationals. In this section,we consider the use of Gauss elimination in the setting of rational arithmetic. Of course in some sense, floating-point *is* rational arithmetic but it uses a very special subset of the rationals and does not comply with the axioms of conventional rational arithmetic. We are concerned here with matrices with entries in the field $\mathbf{Q}$ of rational numbers. The arithmetic operations will be similar in nature to those of Algorithm 15 for back substitution in the integer GE solution of a linear system.

There are choices to be made over the way in which rational numbers are to be stored and manipulated within the computer. The conceptually simplest option is simply to store $q \in \mathbf{Q}$ as a pair of integers representing its numerator and (positive) denominator in its maximally reduced form. Alternatives that have been extensively researched include the use of continued fraction representations and, in particular, the lexicographic continued

fraction, or LCF, arithmetic of Kornerup and Matula [8]. We shall only consider here the $[n/m]$ form in which a rational number is represented as a quotient of two co-prime integers

$$q = \frac{n}{m}$$

where $n, m \in \mathbf{Z}$ have no common factors and $m > 0$. Arithmetic operations are then defined according to the usual rules of rational arithmetic with reduction to this "normalized" form after each arithmetic operation.

It is immediately apparent that many of the range growth problems which plague integer GE will reappear here with comparable severity. Of course the divisions which are inherent in the basic forms of GE and LU factorization can be performed here and restrict the range of values of the rational numbers being represented - *but* these divisions do not necessarily reduce the range of integers needed for the numerators and denominators separately. The chief virtue that would be derived here is the elimination of any rounding errors and therefore definitive answers to questions such as the rank of the matrix and exact values for the determinant and for the solution vector of a linear system.

The algorithm for performing Gauss elimination with a rational matrix can be any variant of Algorithm 1 with real arithmetic operations replaced by rational arithmetic. There is no benefit in detailing this algorithm explicitly. A more detailed discussion in terms of both complexity and the integer range growth for rational Gauss elimination is included in [17]. In this section, we content ourselves with a summary of some relevant results.

Complexity. The most obvious increase in complexity arises out of the mere fact that it is performing rational arithmetic and so every arithmetic operation entails manipulation of both the numerator and denominator. There is also the further complication arising from the reduction of each ordered pair to represent an irreducible fraction. This requires either that each integer is stored as a product of its prime factors, or more reasonably, that the Euclidean algorithm for finding the gcd of two integers is employed and followed with two integer divisions. Because the Euclidean algorithm is iterative we cannot determine the number of integer mod operations that are required. (Also any bounds which could be derived would be hopelessly pessimistic.)

In Table 5, we list the numbers of conventional integer arithmetic operations together with the number of gcd's that are needed. Typically we might expect that a gcd would be equivalent to several integer divisions and that these divisions are, in turn, equivalent to several multiplications. However it should also be noted that, because of the range growth, the multiplications may involve long wordlength integers. On the other hand we may expect the gcd to be much smaller so that the divisor wordlengths may be less extreme. In the operation counts in Table 5, no attempt is made to account for dynamic range considerations or the relative weights to be given to the different operations.

TABLE 5 Integer arithmetic operation counts for the rational GE for solution of an $n \times n$ linear system $A\mathbf{x} = \mathbf{b}$.

Operation	Matrix Factorization	Right-hand side	TOTAL
+ or −	$\frac{1}{3}n\left(n^2-1\right)$	$\frac{1}{2}n\left(n-1\right)$	$\frac{1}{6}n\left(n-1\right)\left(2n+5\right)$
×	$2n^2\left(n-1\right)$	$3n\left(n-1\right)$	$n\left(n-1\right)\left(2n+3\right)$
/	$\frac{2}{3}n\left(n^2-1\right)$	$n(n-1)$	$\frac{1}{3}n\left(n-1\right)\left(2n+5\right)$
gcd	$\frac{1}{3}n\left(n^2-1\right)$	$\frac{1}{2}n(n-1)$	$\frac{1}{6}n\left(n-1\right)\left(2n+5\right)$

Again the biggest single effect is that the number of divisions has increased to $O(n^3)$ in addition to the $O(n^3)$ gcd operations. This time the number of multiplications has also increased by a substantial factor.

Growth in dynamic range. In attempting to analyse the potential growth in the necessary dynamic range for rational GE, we cannot assume any useful reduction in the rational quantities other than that which is a necessary consequence of the elimination procedure itself. For simplicity in this setting, we make the assumption that the original matrix A is in fact an *integer* matrix. This allows us to compare the results of the rational algorithm with those that would be obtained using the divisionless integer algorithm. In much the same way as was described for the fraction-free Algorithm 6, this comparison reveals certain common factors which will be removed by the various reduction steps. This has the effect of reducing the potential range growth in a similar manner to that which we observed in Algorithm 6. Indeed the dynamic range required for the rational algorithm turns out to be identical to that for Algorithm 6. Therefore if the initial matrix is an integer matrix, using rational arithmetic yields no benefit relative to the fraction-free algorithm. If the original matrix consists of rational entries, the range growth problem becomes potentially even more critical since, the "cross-multiplications" needed for the elimination do not appear to generate any obvious and general common factors.

5.3. General rings. Finally, we consider briefly the applicability (and application) of GE in a more general algebraic setting. We are interested here in matrices with entries in a general ring $\mathcal{R}$. There are at least two fundamental questions which arise immediately:

"When do the problems have solutions?" and

"When do the algorithms make sense?"

If $\mathcal{R}$ is a *unique factorization domain* (UFD) then the algorithms described earlier for integer computation remain valid while any rational arithmetic algorithms have obvious analogues in the *field of fractions* $\mathcal{Q}$. For details of the definitions and properties of the various algebraic structures see [4] or any standard text on abstract algebra. In particular this means that the above algorithms have direct analogues for rings of polynomials over $\mathbf{R}$ and $\mathbf{C}$ or their fields of fractions which consist of rational functions over these fields. Some of the specific problems have solutions in slightly more general settings than just a UFD. However the fraction-free algorithms only make sense in a setting where removal of common factors can be achieved. A unique factorization domain is an appropriate setting for this.

It is then the case that all the algorithms described earlier for the integer carry over in the natural way to such a ring: all "arithmetic" being replaced by its corresponding ring operations with division being understood to mean removal of common factors. The resulting algorithms would therefore be much like the integer algorithms in a setting where integers are represented by a list of their prime factors. Removal of common factors would then literally mean their removal from the corresponding lists.

The most interesting settings for these more general algorithms would be the polynomial rings $\mathbf{R}[x]$ and $\mathbf{C}[x]$ for which more complicated fraction-free GE algorithms are built into Computer Algebra Systems such as Maple. The similarity of the algorithms to those for the integers are so great that they are not detailed separately here.

6. CONCLUSIONS

In this paper we have presented a simplified form of Gauss elimination for fraction-free integer computation. This algorithm has several important advantages over conventional

approaches:

It requires no searching for common factors — The factors which are removed are entirely predictable in the light of comparison between integer GE and its standard real arithmetic counterpart.

The dynamic range growth problem is substantially alleviated by the removal of these known factors.

The diagonal of the final upper triangular matrix consists of the determinants of the principal minors of the original matrix. This has the benefit of making positive definiteness (as well as the determinant) easily identifiable.

The range growth that is needed is no worse than that which is inherent in the original matrix in the sense that determinants of all its principal minors will inevitably need to be representable.

The algorithm generalizes in a completely obvious way to more general Unique Factorization Domains — including rings of real or complex polynomials for example.

Acknowledgements This work was supported by grants from the Office of Naval Research through the Naval Air Warfare Center, Aircraft Division, Warminster, PA and the Naval Academy Research Council. The author is also grateful to Bob Williams and Ron Gleeson for several useful discussions related to this work at NAWC, Warminster.

REFERENCES

[1] H.Anton, *Linear Algebra* 4th Ed, Wiley, New York, 1984

[2] N.R.Blachman and M.J.Mossinghoff, *Maple V Quick Reference*, Brooks/ Cole, Pacific Grove, CA, 1994

[3] J.L.Buchanan and P.R.Turner, *Numerical Methods and Analysis*, McGraw-Hill, New York, 1992

[4] D.S.Dummit and R.M.Foote, *Abstract Algebra*, Prentice-Hall, 1991

[5] G.H.Golub and C.F.van Loan, *Matrix Computations* 2nd Ed, Johns Hopkins Press, Baltimore, 1989

[6] B.J.Kirsch and P.R.Turner, *Adaptive beamforming using RNS arithmetic*, Proc ARITH11, IEEE Computer Society, Washington, DC, 1993, pp36-43

[7] B.J.Kirsch and P.R.Turner, *Modified Gauss elimination for adaptive beamforming using complex RNS arithmetic*, Naval Air Warfare Center, Warminster Report NAWCADWAR 94112-50, 1995

[8] P.Kornerup and D.W.Matula, *An on-line arithmetic for bit-pipelined rational arithmetic*, J Parallel and Dist Comp 5 (1989) 310-330

[9] The MathWorks, Inc., *The Student Edition of MATLAB Version 4, User's guide*, Prentice-Hall, Englewood Cliffs, NJ, 1995

[10] J.J.Modi, *Parallel Algorithms and Matrix Computations*, Oxford University Press, Oxford, 1988

[11] J.M.Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum, New York, 1988

[12] N.R.Scott, *Computer Number Systems and Arithmetic*, Prentice-Hall, 1985

[13] M.A.Soderstrand, W.K.Jenkins, G.A.Jullien and F.J.Taylor, *Residue Number System Arithmetic: Modern applications in digital signal processing*, IEEE, New York, 1986

[14] P.H.Sterbenz, *Floating-point computation*, Prentice-Hall, 1974

[15] N.Szabo and R.Tanaka, *Residue Arithmetic and its Application to Computer Technology*, McGraw-Hill, 1967

[16] P.R.Turner and B.J.Kirsch, *Operation complexity for integer or RNS Gaussian elimination*, Report NAWCADWAR - 95004-4.5, 1995

[17] P.R.Turner, *Gauss elimination: Workhorse of linear algebra*, NAWC-AD Tech Rep 1996

[18] P.R.Turner, *Low rank determination using least squares*, NAWC-AD Tech Rep 1996

[19] J.H.Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, 1965

[20] S.Wolfram, *Mathematica* 2nd Ed, Addison-Wesley, New York, 1991

Distribution List

Report No. NAWCADPAX- -96 - 196 - TR

	No. of Copies
Office of Naval Research	2
800 N. Quincy St.	
Arlington, VA 22217	
Marine Corps Research Center	
2040 Broadway Street	
Quantico, VA 22134-5107	
Marine Corps University Libraries	2
Naval Air Systems Command	
Air-5002	
Washington, DC 20641-5004	
Technical Information & Reference Center	2
Naval Air Warfare Center, Aircraft Division	
Building 407	
Patuxent River, MD 20670-5407	
Naval Air Station Central Library	2
Naval Air Systems Command (NAVAIR)	
Jefferson Plaza Bldg 1., 1421 Jefferson Davis Hwy	
Arlington, VA 2243-5120	
Director Science & Technology (4.0T)	2
Naval Sea Systems Command	
2531 Jefferson Davis Hwy	
Arlington, VA 22242-5100	
Technical Library, (SEA04TD2L)	2
Defense Technical Information Center	
Cameron Station BG5	
Alexandria, VA 22304-6145	
DTIC-FDAB	2
U.S. Naval Academy	
Annapolis, MD 21402-5029	
Peter R. Turner (Mathematics Department)	10
Dr. Richard Werking (Nimitz Library)	2
Naval Air Warfare Center	
Weapons Division	
China Lake, CA 93555-6001	
Head Research & Tech. Div. (NAWCWPNS-474000D)	2
Computational Sciences (NAWCWPNS-474400D)	2
Mary-Deirdre Coraggio (Library Division, C643)	2

Distribution List, cont.

Report No. NAWCADPAX- -96- 196 - TR

	No. of Copies
Naval Postgraduate School Monterey, CA 93943-5002	
Dudley Knox Library	2
Naval Research Laboratory(NRL) 4555 Overlook Ave, SW Washington, DC 20375-5000	
Center for Computational Science (NRL-5590)	2
Superint., Lab. for Comput. Phy & Fluid Dynamics (NRL-6400)	2
Ruth H. Hooker Research Library (5220)	2
Naval Command, Control & Ocean Surveillance Center 200 Catalina Blvd San Diego, CA 92147-5042	
Technical Library (NRAD-0274)	2
Signals Warfare Div (NRAD-77)	2
Analysis & Simulation Div. (NRAD-78)	2
Director of Navigation & Air C3 Dept. (NCCOSC-30)	2
Naval Air Warfare Center Aircraft Division Warminster Warminster, PA 18974-0591	
Warfare Planning Systems (4.5.2.1.00R07)	2
Tactical Inf. Systems (4.5.2.2.00R07)	2
Mission Comp. Processors (4.5.5.1.00R07)	2
Dr. Robert M. Williams (4.5.5.1.00R07)	20
Acoustic Sensors (4.5.5.4.00R07)	2
RF Sensors (4.5.5.5.00R07)	2
EO Sensors (4.5.5.6.00R07)	2
Inductive Analysis Branch (4.10.2.00R86)	2
TACAIR Analysis Division (4.10.1.00R86)	2
Operations Research Analysis Branch (4.10.1.00R86)	2
Advanced Concepts Branch (4.10.3.00R86)	2
Nav. Aval. Sys. Dev. Division (3.1.0.9)	2
Anthony Passamante (4.5.5.3.4.00R07)	2
Elect. Systems BR (4.8.2.2.00R08)	2
Dr. Richard Llorens (4.3.2.1.00R08)	2
Advanced Processors (4.5.5.1.00R07)	2
Mission & Sensors Integrations (4.5.5.3.000R07)	2
Applied Signal Process BR (4.5.5.3.4.00R07)	2